SET09107: Advanced Database Systems

2019/20 Coursework

Name: Amelia Handley

Matriculation Number: 40326169

1. Task 1

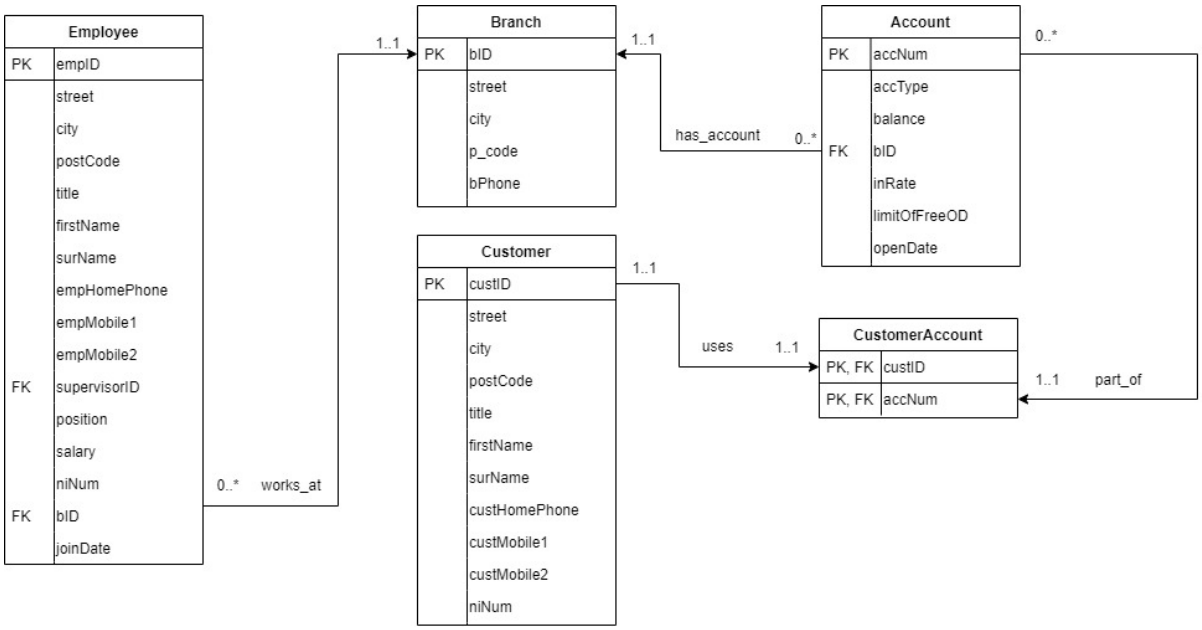

Image 1: An Entity Relational Diagram corresponding to the relational database schema for the bank

2. Task 2

Object-relational models are an extension of relational models by supporting features such as queries and fast commits [1] but also combines features from object-oriented models such as objects, collections, classes and inheritances. This meant re-designing the database to make use of these object-relational features tables to capture more of the semantics of the bank application without losing semantics of the relational database schema (see Image 2).

2.1. Design

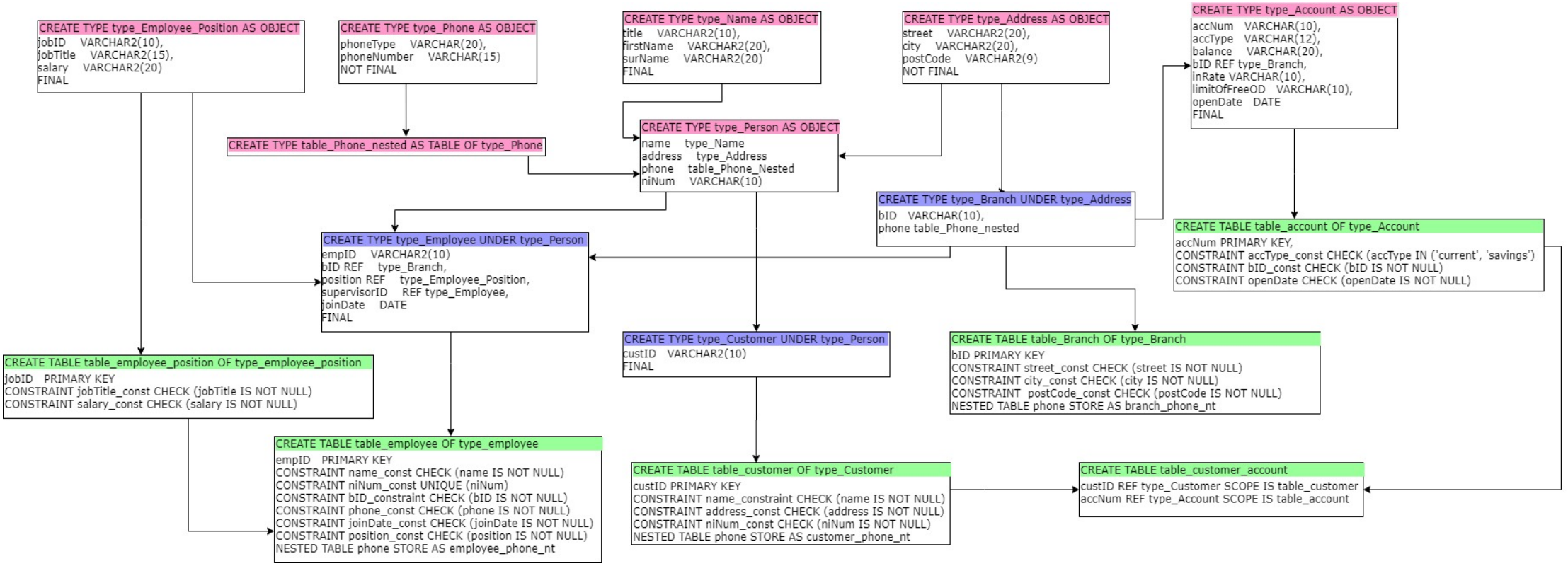

Image 2: Object-Relational Diagram of the Bank Schema

2.1.1. Structured Types

To capture these semantics object-types were used to serve as blueprints to define both the structure and the behaviour of the database [1]. They contain one or more named attributes, each having their own data type. The attributes are properties that are used to describe the instance of a type [2].

In the case of the Bank Schema, initially four object-types were required. These included:

- **Type_name**: holds the information for a person's title (e.g. Mr, Mrs, Ms, Miss) [3], a person's first name and a person's surname,
- **Type_address**: holds the information for an address including the street name, the city and the post code,
- **Type_phone**: holds the information for the type of phone that is being entered (e.g. a branch phone, mobile phone or a home phone) and the number for that phone,
- **Type_person**: gathers the information from the three previous types to be used as composite attributes (e.g. the name, address as well as phone numbers) which would be associated to a person. Additionally, the attribute to store a national insurance number has been added.

For type address, phone and person the "NOT FINAL" statement was included to allow the creation of subtypes of each of the types [1]. These object-types are known as supertypes. Whereas for the type name, the "FINAL" statement was used so no subtypes could be derived from it.

In addition to the supertypes created, two object-types were created. These were:

- **Type_employee_position**: contains a job identification number, a job title variable (e.g. manager, accountant etc) and the salary a person is paid in their position,
- **Type_account**: contains an account number, an account type (e.g. savings or a current account), the balance of the account, the branch which the account was opened, the in rate of the account, the limit of free overdraft and the date which the account was created.

As these types were not to be inherited from, the "FINAL" statement was included.

2.1.2. Inheritance

Within object-relational models a key concept is inheritance [4]. Object types within Oracle support inheritance which allows another type or the subtypes to be created from the previous type (i.e. supertype). Additionally, inherited attributes will contain new attributes as well as those that they have inherited [3]. Inheritance prevents the need to reuse code from

supertypes to subtypes as they are included, reducing the need to store the same data in multiple locations. To declare a subtype “UNDER” is used to enable it to inherit from the supertype.

In the case of the re-design, there were three subtypes created:

- **Type_branch**: is a subtype of type_address meaning it inherits attributes for the address as well as having branch identification number and will also contain the phone number of the branch,
- **Type_employee**: is a subtype of type_person meaning it collects a persons name, address, phone numbers and national insurance number. Also, it collects the employee identification number, the branch the employee works at, their position in the branch, the supervisor identification number and the date the employee joined the bank,
- **Type_customer**: is a subtype of type_person meaning it inherits a persons name, address, phone numbers and national insurance number. Additionally, it adds a customer identification number.

Since there are to be no subtypes created from the subtypes detailed above, the key word “FINAL” was used.

The final application of the types was to make tables for a customer account, an employee position, an employee, a customer and a branch, each with different attributes specified by the types they have inherited and their additional features (e.g. the customer identification number).

2.1.3. References

A reference is a pointer to an instance of an object. References allow for tuples to have direct references. So, instead of using the primary key to connect the tables and types, the reference can be used instead [5].

For the re-design of the bank schema, references were used within two of the object-types:

- **Type_employee**: to get the branch information an employee works at a reference to the branch type was used as well as to the employee position table which gets the job title of the employee and their salary. Additionally, a reference was made to the supervisor which uses the employee type, as a supervisor will require all the information within the type,

- **Type_account**: For an account, a reference is needed to the branch type which will collect the branch information (such as the branch identification, address and phone number),

It was also used within one of the tables:

- **Table_customer_account**: Customer account is an important table which connects the account table and the customer table. It does this by referencing the account table from the account number and to the customer table from the customer table. Additionally, the command “SCOPE IS” was used to gather the information defined by the references at compile time [6].

2.1.4. Methods

Object Methods are functions that can be declared within an object type in oracle. They can be used to define behaviour on objects of the desired type you want to perform [3]. The member method is defined in the object type or can be altered after their creation using the “ALTER TYPE” command. They are defined using “MEMBER FUNCTION”.

In the case of this project, member functions were implemented for two of the types: type_person and type_branch.

```
1  ALTER TYPE type_Branch
2  ADD MEMBER FUNCTION print_address RETURN VARCHAR2 CASCADE;
3  /
4  CREATE OR REPLACE TYPE BODY type_Branch AS
5  MEMBER FUNCTION print_address RETURN VARCHAR2 IS
6  BEGIN
7      RETURN SELF.street|| ', '|| SELF.city|| ', ' || SELF.postCode;
8  END print_address;
9  END;
10 /
```

Image 3: Member function for type_branch.

To illustrate, within the type_branch the print_address member function was created to return the full address without having to access each individual part in the select queries (i.e. the street, city and the postcode) (Image 3). The SELF parameter is used to denote the object instance that is currently invoking the method [7].

A member method was also used in type_person to return a person’s full name, including their title, first name and surname and to return the address (a similar function to Image 3).

2.1.5. Constraints

A constraint is used to define an integrity constraint, meaning that it can restrict the value inputted into the database. Within oracle, six types of constraints can be used when making the tables for the database [8].

In the context of this project, the following 5 were used:

- **NOT NULL**: it requires a database value for the attribute, for example someone's name,
- **UNIQUE**: means that no two or more rows of the database can have the same value, for example the national insurance number of a customer or a colleague,
- **PRIMARY KEY**: within each table a primary key was created. This contains a combination of the NOT NULL constraint and the UNIQUE constraints within one declaration. An example of this in the database would be the branch identification number. This is used to identity and access the branch information from another table such as the employee table.
- **CHECK**: requires the value to meet a specified condition, for example having either a 'savings' or 'current' account,
- **REF**: describes the relationship between the REF and the object that it is referenced to [8] therefore, eliminating the need for a JOIN function to connect tables.

2.1.6. Collections

Collections are a group of ordered elements that are of the same type. It is a concept that can be used in datatypes such as arrays and nested tables.

Nested tables, as their name suggests, are tables that are in tables. This was chosen as a suitable approach as there were different types of phone being used to store information in the database. Nested tables are single-dimensional, meaning that each row has a single column of data, acting like a one-dimensional array [9]. This means that since nested tables can condense multiple columns, such as the phone type and the phone number into a single column without losing any of their data [10].

For the instance of the project, a nested table was used from type phone.

```
1  CREATE TYPE type_Phone AS OBJECT
2      (phoneType VARCHAR(20),
3      phoneNumber VARCHAR(15))
4      NOT FINAL
5  /
6  CREATE TYPE table_Phone_nested AS TABLE OF type_Phone
7  /
```

Image 4: Nested table for phones.

For example, there was a branch phone, home phone and multiple mobile phone numbers meaning that there were occasions where one phone type was being used, and others where multiple were. This means that the size of the data is unknown which is no issue as the size of the nested table does not need to be predetermined. [10].

2.2. Alternative Possible Object-Relational Representations

2.2.1. References

A **value** is a type of reference which takes it argument of a correlation variable, which is associated to an object, and returns the object instances stored within a table [11]. However, they are not ideal to use when returning by a pointer or a class [12]. Instead a reference (REF()) was used.

A dereference operator (or **deref**) returns the value of an object, so unlike values it takes a correlation argument. So, if you want to retrieve the target object instead of the reference to it, a deref can be used [13]. However, the information for the database could be accessed using the REF() command to return the object ID within the oracle database [14].

2.2.2 Collections

A **variable sized array** (or a varray) is a collection data type that could have been used during this coursework. They are like nested tables; Varrays have advantages over nested tables as they occupy less space, as their upper bound limit (or space capacity) must be declared in the CREATE TYPE statement. Additionally, as a varray is stored in the other columns in a row, unlike nested tables which are stored in a separate table, accessing the information is faster. However, they must know the number of elements being inserted into the varray in advance which was not suitable for the phone table as it was unknown how many types and numbers would be added [10].

Task 4

4.A. Find employees whose first names includes the string “st” and live in Edinburgh.

SQL Statement for 4.A:

```
SELECT  e.print_name() AS "Employee Name"
FROM  table_employee e
WHERE lower(e.name.firstName) LIKE lower('%st%')
AND e.address.city = 'Edinburgh';
```

	Employee Name
1	Mr Stuart Jones
2	Miss Stacey Yale
3	Mr Stanley Burns

Image 5. Output from SQL Statement A (Image 1)

Using the member method print_name() the employees full name is taken from the employee table. There is then a where query to search for specifically “st” within the first name of the employee as well as a query to find employees in Edinburgh.

4.B. Find the number of savings accounts at each branch

SQL Statement for 4.B:

```
SELECT b.bID.bID AS "Branch ID",
         b.bID.print_address() AS "Branch Address",
         COUNT(b.accType) AS "Number of Savings Accounts"
FROM table_account b
WHERE accType = 'savings'
GROUP BY b.bID.bID, b.bID.print_address()
ORDER BY b.bID.bID ASC;
```

	Branch ID	Branch Address	Number of Savings Accounts
1	901	Market, Edinburgh, EH1 5AB	1
2	905	Princes St, Edinburgh, EH4 8HY	3
3	907	Cross, London, N7C 7BD	1
4	910	George St, Edinburgh, EH4 6YG	4
5	911	Chelsea, London, N8C 0LN	1
6	920	Bear, London, N6U 9QW	1

Image 6. Output from SQL Statement B

Using access component attributes (i.e. the dot ‘.’ notation), the select statement looks for the branch ID from the account table. Also, the member method print_address() was used to get the address of the bank. The count method is used to count the types of accounts [15]. The group by clause to group rows of the same value, which in this instance would be the

branch ID, the addresses of each of the banks and the account type which would be a savings, which is specified in the where clause.

4.C. At each branch, find the customers who have the highest balance in their savings account

SQL Statement for 4.C. [16]:

```
SELECT c.accNum.bID.bID AS "Branch ID",
         c.custID.print_name() AS "Customer Name",
         c.accNum.balance AS "Savings Balance"
FROM   table_customer_account c,
       (SELECT a.bID.bID AS bID,
               a.accType AS accType,
               MAX(a.balance) AS balance
        FROM table_account a
        GROUP BY a.bID.bID, a.accType) highestBalance
WHERE c.accNum.bID.bID = highestBalance.bID
AND c.accNum.balance = highestBalance.balance
AND c.accNum.accType = highestBalance.accType
AND c.accNum.accType = 'savings'
ORDER BY c.accNum.bID.bID ASC;
```

	Branch ID	Customer Name	Savings Balance
1	901	Mr Arthur Dayne	567.01
2	905	Mr Zach Martin	8945.89
3	907	Mr Brian Davis	12077.90
4	910	Miss Mercedes Wilson	13567.05
5	911	Mrs Bella Scott	14555.09
6	920	Mrs Maria Musk	25604.24

Image 7. Output from SQL Statement C

The select query gathers the branch identification number from where the account was opened, the customers name and the balance of the account. Within the from query, another subquery(or inline view), was used to find the highest balance of each account at each branch in the table_account [17]. This was to ensure that only one balance (the highest balance) [18] was returned from each branch and not all the balances for each account at the branches. Within the where clause the branch identification from the table_customer_account is set to equal the fields aliased in the “highestBalance” from the subquery (i.e. the branches that were returned from the subquery statement). The where statement also connects the balance from the customer account table and the account type to the figures returned from the highest balance. Additionally, there is a where query that ensures only the savings accounts are displayed.

4.D. Find employees who are supervised by a manager and have accounts in the bank

SQL Statement for 4.D:

```
SELECT e.bID.print_address() AS "Work Address",
       c.accNum.bID.print_address() AS "Customer Branch Address",
         e.print_name() AS "Employee Name"
FROM table_employee e, table_customer_account c
WHERE c.custID.print_name() = e.print_name()
AND e.supervisorID.position.jobTitle = 'Manager'
ORDER BY e.bID ASC;
```

	Work Address	Customer Branch Address	Employee Name
1	Bear, London, N6U 9QW	Chelsea, London, N8C 0LN	Mrs Bella Scott
2	Market, Edinburgh, EH1 5AB	Princes St, Edinburgh, EH4 8HY	Mr Homer Simpson
3	Bridge, Glasgow, G18 1QQ	Princes St, Edinburgh, EH4 8HY	Ms Rose Tate

Image 8. Output from SQL Statement B

The select query searches the employee table for the employee name and the address the employee works at. It also looks for the account number from the table_customer_account to find the branch address the employee made an account with (as a customer). The where clause ensures that the name in the employee table is the same found in the customer account table as well as looking for employees whose supervisor is a manager.

4.E. At each branch, find customers who have the highest free overdraft limit in all current accounts that are joint accounts

SQL Statement for 4.E:

```
SELECT c.accNum.bID.bID AS "Branch ID",
       c.accNum AS "Customer Account Number",
         c.accNum.limitOfFreeOD AS "Overdraft Limit"
FROM   table_customer_account c,
       (SELECT a.bID.bID AS bID,
                  MAX(a.limitOfFreeOD) AS odLimit
         FROM table_account a
         GROUP BY a.bID.bID) freeOD
WHERE c.accNum.limitOfFreeOD = freeOD.odLimit
AND c.accNum.bID.bID = freeOD.bID
AND c.accNum.accType = 'current'
GROUP BY c.accNum.bID.bID, c.custID.name.surName, c.accNum.limitOfFreeOD
HAVING COUNT(c.accNum) = 2
ORDER BY c.accNum.bID.bID ASC;
```

	Branch ID	Customer Account Number	Overdraft Limit
1	911	A1786	300
2	918	A1345	800

Image 9. Output from SQL Statement E

The select statement gathers the branch identification number, the customer account numbers and the overdraft limit that the account has. A subquery is used to find the highest free overdraft limit in the table_account [17]. This will find the account with the highest free overdraft limit at each branch. The where clause connects the table_customer_account with the results from the subquery “freeOD” on the overdraft limit and the branch identification. In addition, it will only display the accounts that are a current account type. Finally, since the query is finding joint accounts the HAVING COUNT function [16] is used to find account numbers in the customer account table which have two customers associated to it.

4.F. Find customers who have more than one mobile, and at least one of the numbers starts with 0750

SQL Statement for 4.F.:

```
SELECT c.custID AS "Customer ID"
       t.phoneType AS "Phone Type",
       t.phoneNumber AS "Mobile Number"
FROM table_customer c, table(c.phone) t
WHERE t.phoneType LIKE '%Mobile2%'
AND t.phoneNumber LIKE '%0750%'
GROUP BY c.custID, t.phoneType, t.phoneNumber;
```

	Customer ID	Phone Type	Mobile Number
1	Cust103	Mobile2	07504536389
2	Cust108	Mobile2	07504554321
3	Cust121	Mobile2	0750937837

Image 10. Output from SQL Statement F

The select query collects the values from the customer table for the customer’s name and from the nested phone table, the type of phone (i.e. home phone, mobile 1, mobile 2) and the number associated with the phone. The where clause finds the customers with a second mobile device and their mobile device begins with a “0750”.

4.G. Find the number of employees who are supervised by Mrs Smith, who is supervised by Mr Jones

SQL Statement for 4.G.:

```
SELECT e.supervisorID.print_name() AS "Supervisor",
       COUNT(e.empID) AS "Number of Employees"
FROM table_employee e
WHERE  e.supervisorID.name.title  = 'Mrs'
AND e.supervisorID.name.surName = 'Smith'
OR e.supervisorID.name.title = 'Mr'
AND e.supervisorID.name.surName = 'Jones'
GROUP BY e.supervisorID.print_name()
```

	Supervisor	Number of Employees
1	Mrs Alison Smith	2
2	Mr Stuart Jones	1

Image 11. Output from SQL Statement G

In the select query the supervisor name is found in the table_employee and there is a COUNT function used to count the number of employees by their employee identification number. There is then a where query which looks for supervisors either ‘Mrs Smith’ or ‘Mr Jones’ [19]. So, the result is the number of employees under Mrs Smith as 2 and the employees under Mr Jones as 1 (i.e. Mrs Smith).

Task 5.

5.1. Entity Relational Models

Entity-Relational Models contain the data and relationships represented by a collection of inter-related tables. They are often used to represent real life scenarios as entities so because of this it is often called a top-down approach to database design. It does this by storing the data defined as tables which hold specific information. Connections between the entities are shown as relationships [20].

This is an advantage of an ER model is that the data requirements are easily understandable using clear diagrams such as Image 1. It is easy to see the relationship between the defined entities on the diagram for instance, each employee works at a branch or every account must be associated to a customer through the customer account table.

However, E-R models only store data meaning that methods to retrieve data, such as member functions to retrieve a person’s name (the print_name function), cannot be used. Additionally, it cannot expressed heterogeneous collections well as it does not map well to the table, for example it would not be able to implement the phone nested table so you must enter the specific phone types and numbers for each of the attributes when creating the table.

5.2. Object-Relational Models

Object-relational model is like the E-R model however it extends on it with concepts such as encapsulation, methods and object identity. This means that data is represented in the form of objects. Objects with similar functionalities or behaviours are grouped together and linked to different other objects. For instance, in the re-design of the banking scenario attributes are gathered to make a person type based on other attributes such as name, address and phone numbers. This is then used to make employee and customer subtypes [21].

This method of inheritance is an advantage to the O-R model as it means that the data types can be reused in different objects, reducing the cost of having data of the same value in multiple locations. An example of this would be the nested table for phone numbers. It can handle larger and complex data types compared to E-R models, such as references and methods [20].

Additionally, a key feature of the object-relational model is that is it able to write and execute methods unlike the E-R model. An example of a method is the print_address() function used within the type_branch. This method enables the street name, the city and the post of the branch to printed into the same column when it is executed instead of having to get each of the attributes in separate columns.

Task 6.

Image 19 illustrates how to remove all of the tables and types implemented for the bank database in a series of drop statements.

```
--DROP TYPES

DROP TYPE TYPE_ACCOUNT FORCE;
DROP TYPE TYPE_ADDRESS FORCE;
DROP TYPE TYPE_BRANCH FORCE;
DROP TYPE TYPE_CUSTOMER FORCE;
DROP TYPE TYPE_EMPLOYEE FORCE;
DROP TYPE TYPE_NAME FORCE;
DROP TYPE TYPE_PERSON FORCE;
DROP TYPE TYPE_PHONE FORCE;
DROP TYPE TYPE_EMPLOYEE_POSITION FORCE;
DROP TYPE TABLE_PHONE_NESTED FORCE;

--DROP TABLES

DROP TABLE TABLE_ACCOUNT;
DROP TABLE TABLE_EMPLOYEE;
DROP TABLE TABLE_CUSTOMER;
DROP TABLE TABLE_CUSTOMER_ACCOUNT;
DROP TABLE TABLE_EMPLOYEE_POSITION;
DROP TABLE TABLE_BRANCH;
```

Image 12: Dropping all types and tables from the database.

References

[1] “1.3 Key Features of the Object-Relational Model,” [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/adobj/key-features-object-relational-model.html#GUID-8A38BA82-CCD5-4424-AE19-10A994E25B0E. [Accessed 18 March 2020].

[2] “Structured Types,” IBM, [Online]. Available: https://www.ibm.com/support/knowledgecenter/vi/SSEPGG_9.7.0/com.ibm.db2.luw.admin.structypes.doc/doc/c0006441.html. [Accessed 19 March 2020].

[3] K. Castro, “Object Relational Data Model,” 25 July 2018. [Online]. Available: https://www.tutorialspoint.com/Object-relational-Data-Model.

[4] C. G. Aksakalli, “Mapping Inheritance to Relational Databases,” 16 June 2015. [Online]. Available: https://aksakalli.github.io/2015/06/16/mapping-inheritance-to-relational-databases.html.

[5] “Object-Relational Database Systems,” [Online]. Available: http://www.cburch.com/cs/340/reading/ordbms/index.html.

[6] “PL/Scope in Oracle Database 11g Release 1 (11.1),” Oracle Base, [Online]. Available: https://oracle-base.com/articles/11g/plscope-11gr1. [Accessed 10 March 2020].

[7] “2.2. Object Methods,” Oracle, [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/20/adobj/object-methods.html#GUID-0A7901B9-EFB4-4243-B01B-2325A3706F6E. [Accessed 15 March 2020].

[8] “Constraint,” Oracle, [Online]. Available: https://docs.oracle.com/cd/B19306_01/server.102/b14200/clauses002.htm. [Accessed 17 March 2020].

[9] “PL/SQL Collections and Records,” Oracle, [Online]. Available: https://docs.oracle.com/cd/A97630_01/appdev.920/a96624/05_colls.htm. [Accessed 20 March 2020].

[10] “Oracle,” Using PL/SQL Collections and Records, [Online]. Available: https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/collections.htm#LNPLS00503. [Accessed 17 March 2020].

[11] “Database SQL Reference: Value,” Oracle, [Online]. Available: https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions209.htm. [Accessed 20 March 2020].

[12] “7.4a — Returning values by value, reference, and address,” LearnCpp, 23 January 2020. [Online]. Available: https://www.learncpp.com/cpp-tutorial/74a-returning-values-by-value-reference-and-address/. [Accessed 20 March 2020].

[13] [Online]. Available: https://www.relationaldbdesign.com/oracle-pl-sql-programming/module3/writing-deref-queries.php. [Accessed 19 March 2020].

[14] “DEREF,” Oracle, [Online]. Available: https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions045.htm. [Accessed 20 March 2020].

[15] “SQL: COUNT Function,” Tech on the Net, [Online]. Available: https://www.techonthenet.com/sql/count.php. [Accessed 7 March 2020].

[16] “Oracle/PLSQL: HAVING Clause,” [Online]. Available: https://www.techonthenet.com/oracle/having.php. [Accessed 8 March 2020].

[17] “Oracle/PLSQL: Subqueries,” [Online]. Available: https://www.techonthenet.com/oracle/subqueries.php. [Accessed 8 March 2020].

[18] “SQL: Max Function,” [Online]. Available: https://www.techonthenet.com/sql/max.php. [Accessed 10 March 2020].

[19] “Oracle/PLSQL: Combining the AND and OR Conditions,” [Online]. Available: https://www.techonthenet.com/oracle/and_or.php. [Accessed 8 March 2020].

[20] “Difference between RDBMS and OODBMS,” Geeks for Geeks, [Online]. Available: https://www.geeksforgeeks.org/difference-between-rdbms-and-oodbms/. [Accessed 18 March 2020].

[2 A. Onsman, “Comparison between E-R Model and Object Oriented Model,” 24 July 2018.
1] [Online]. Available: https://www.tutorialspoint.com/Comparison-between-E-R-Model-and-Object-Oriented-Model. [Accessed 18 March 2020].